In [1]:

```python
import os
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
import torch
import timm
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.optim as optim
from PIL import Image
```

In [2]:

```python
# Paths to datasets
train_dir = '/kaggle/input/retinal-disease-classification/Training_Set/Trainir
val_dir = '/kaggle/input/retinal-disease-classification/Evaluation_Set/Evaluat
test_dir = '/kaggle/input/retinal-disease-classification/Test_Set/Test_Set/Tes

train_labels_path = '/kaggle/input/retinal-disease-classification/Training_Set
val_labels_path = '/kaggle/input/retinal-disease-classification/Evaluation_Set
test_labels_path = '/kaggle/input/retinal-disease-classification/Test_Set/Test
```

In [3]:

```python
# Load labels
train_labels = pd.read_csv(train_labels_path)
val_labels = pd.read_csv(val_labels_path)
test_labels = pd.read_csv(test_labels_path)
```

In [4]:

```python
# Print dataset shapes
print("Train labels shape:", train_labels.shape)
print("Validation labels shape:", val_labels.shape)
print("Test labels shape:", test_labels.shape)
```

```
Train labels shape: (1920, 47)
Validation labels shape: (640, 47)
Test labels shape: (640, 47)
```

In [5]:

```python
# Data Preprocessing
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),  # Small shifts
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
])
```

In [6]:

```python
class RetinalDataset(Dataset):
```

```python
    def __init__(self, image_dir, labels_df, transform=None):
        self.image_dir = image_dir
        self.labels_df = labels_df
        self.transform = transform

    def __len__(self):
        return len(self.labels_df)

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, f"{self.labels_df.iloc[idx]['I
        image = Image.open(img_path).convert('RGB')
        label = self.labels_df.iloc[idx]['Disease_Risk']
        if self.transform:
            image = self.transform(image)
        return image, torch.tensor(label, dtype=torch.float32)
```

In [7]:

```python
# Create datasets and loaders
train_dataset = RetinalDataset(train_dir, train_labels, transform=transform)
val_dataset = RetinalDataset(val_dir, val_labels, transform=transform)
test_dataset = RetinalDataset(test_dir, test_labels, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

In [8]:

```python
import torch
print("CUDA Available:", torch.cuda.is_available())
print("Device:", torch.device("cuda" if torch.cuda.is_available() else "cpu"))
```

```
CUDA Available: True
Device: cuda
```

In [9]:

```python
# Load pre-trained Vision Transformer (ViT) and Swin Transformer models
vit_model = timm.create_model('beit_base_patch16_224', pretrained=True, num_cl
swin_model = timm.create_model('swin_base_patch4_window7_224', pretrained=True
```

```
model.safetensors:   0%|          | 0.00/350M [00:00<?, ?B/s]
model.safetensors:   0%|          | 0.00/353M [00:00<?, ?B/s]
```

In [10]:

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vit_model = vit_model.to(device)
swin_model = swin_model.to(device)
```

In [11]:

```python
# Loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
vit_optimizer = optim.Adam(vit_model.parameters(), lr=1e-4, weight_decay=1e-5)
swin_optimizer = optim.Adam(swin_model.parameters(), lr=1e-4, weight_decay=1e-
```

In [12]:

```python
def train_model(model, train_loader, val_loader, criterion, optimizer, schedul
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    best_val_loss = float('inf')
```

```
    patience_counter = 0

    # Store losses and accuracies
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_loss, correct, total = 0.0, 0, 0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device).unsqueeze(1)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            preds = (torch.sigmoid(outputs) > 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)

        train_acc = correct / total
        train_loss /= len(train_loader)

        # Validation Phase
        model.eval()
        val_loss, val_correct, val_total = 0.0, 0, 0

        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device).unsqueez
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                preds = (torch.sigmoid(outputs) > 0.5).float()
                val_correct += (preds == labels).sum().item()
                val_total += labels.size(0)

        val_acc = val_correct / val_total
        val_loss /= len(val_loader)

        # Store metrics
        train_losses.append(train_loss)
        val_losses.append(val_loss)
        train_accs.append(train_acc)
        val_accs.append(val_acc)

        # Print Progress
```

```
        print(f"Epoch {epoch+1}/{epochs} | Train Loss: {train_loss:.4f}, Trair

        # Check for Best Model (Early Stopping)
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), "best_model.pth")
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter >= patience:
            print("Early stopping triggered! Stopping training.")
            break

        # Adjust Learning Rate (if scheduler provided)
        if scheduler:
            scheduler.step()

    print("Training complete. Best Validation Loss:", best_val_loss)

    return train_losses, val_losses, train_accs, val_accs
```

In [13]:

```
# Loss function
criterion = nn.BCEWithLogitsLoss()

# Optimizers
vit_optimizer = optim.Adam(vit_model.parameters(), lr=1e-4, weight_decay=1e-5)
swin_optimizer = optim.Adam(swin_model.parameters(), lr=1e-4, weight_decay=1e-

# Learning Rate Scheduler (Reduce LR after 5 epochs)
vit_scheduler = torch.optim.lr_scheduler.StepLR(vit_optimizer, step_size=5, ga
swin_scheduler = torch.optim.lr_scheduler.StepLR(swin_optimizer, step_size=5,


# Train ViT
print("Training ViT Model:")
vit_train_losses, vit_val_losses, vit_train_accs, vit_val_accs = train_model(
    vit_model, train_loader, val_loader, criterion, vit_optimizer, vit_schedul
)

# Train Swin Transformer
print("\nTraining Swin Transformer Model:")
swin_train_losses, swin_val_losses, swin_train_accs, swin_val_accs = train_moc
    swin_model, train_loader, val_loader, criterion, swin_optimizer, swin_sche
)
```

```
Training ViT Model:
Epoch 1/15 | Train Loss: 0.5243, Train Acc: 0.7906 | Val Loss: 0.5174, Val Acc:
0.7906
Epoch 2/15 | Train Loss: 0.4999, Train Acc: 0.7911 | Val Loss: 0.5138, Val Acc:
0.7906
Epoch 3/15 | Train Loss: 0.4936, Train Acc: 0.7911 | Val Loss: 0.4853, Val Acc:
0.7906
Epoch 4/15 | Train Loss: 0.4835, Train Acc: 0.7911 | Val Loss: 0.4699, Val Acc:
0.7906
Epoch 5/15 | Train Loss: 0.4774, Train Acc: 0.7958 | Val Loss: 0.4939, Val Acc:
0.7797
Epoch 6/15 | Train Loss: 0.4609, Train Acc: 0.8000 | Val Loss: 0.4746, Val Acc:
0.7766
Epoch 7/15 | Train Loss: 0.4490, Train Acc: 0.8026 | Val Loss: 0.4512, Val Acc:
0.8109
Epoch 8/15 | Train Loss: 0.4315, Train Acc: 0.8177 | Val Loss: 0.4600, Val Acc:
0.8047
Epoch 9/15 | Train Loss: 0.4254, Train Acc: 0.8193 | Val Loss: 0.4257, Val Acc:
0.8078
Epoch 10/15 | Train Loss: 0.4124, Train Acc: 0.8203 | Val Loss: 0.4004, Val Ac
c: 0.8234
Epoch 11/15 | Train Loss: 0.3945, Train Acc: 0.8344 | Val Loss: 0.3920, Val Ac
c: 0.8297
Epoch 12/15 | Train Loss: 0.3811, Train Acc: 0.8427 | Val Loss: 0.3974, Val Ac
c: 0.8172
Epoch 13/15 | Train Loss: 0.3831, Train Acc: 0.8333 | Val Loss: 0.3694, Val Ac
c: 0.8438
Epoch 14/15 | Train Loss: 0.3765, Train Acc: 0.8417 | Val Loss: 0.4193, Val Ac
c: 0.8141
Epoch 15/15 | Train Loss: 0.3834, Train Acc: 0.8380 | Val Loss: 0.3800, Val Ac
c: 0.8406
Training complete. Best Validation Loss: 0.3694283176213503

Training Swin Transformer Model:
Epoch 1/15 | Train Loss: 0.4216, Train Acc: 0.8130 | Val Loss: 0.3429, Val Acc:
0.8187
Epoch 2/15 | Train Loss: 0.3235, Train Acc: 0.8531 | Val Loss: 0.2737, Val Acc:
0.8703
Epoch 3/15 | Train Loss: 0.2823, Train Acc: 0.8604 | Val Loss: 0.2564, Val Acc:
0.8703
Epoch 4/15 | Train Loss: 0.2519, Train Acc: 0.8745 | Val Loss: 0.3421, Val Acc:
0.8250
Epoch 5/15 | Train Loss: 0.2485, Train Acc: 0.8953 | Val Loss: 0.2488, Val Acc:
0.8875
Epoch 6/15 | Train Loss: 0.2171, Train Acc: 0.9021 | Val Loss: 0.2785, Val Acc:
0.8859
Epoch 7/15 | Train Loss: 0.1908, Train Acc: 0.9109 | Val Loss: 0.3362, Val Acc:
0.8688
Epoch 9/15 | Train Loss: 0.1786, Train Acc: 0.9203 | Val Loss: 0.2269, Val Acc:
0.8875
Epoch 10/15 | Train Loss: 0.1819, Train Acc: 0.9146 | Val Loss: 0.2550, Val Ac
c: 0.8906
Epoch 11/15 | Train Loss: 0.1516, Train Acc: 0.9318 | Val Loss: 0.2597, Val Ac
c: 0.8859
```

```
Epoch 12/15 | Train Loss: 0.1283, Train Acc: 0.9443 | Val Loss: 0.3079, Val Ac
c: 0.8719
Early stopping triggered! Stopping training.
Training complete. Best Validation Loss: 0.22686027251183988
```

In [19]:
```
# Evaluation function
def evaluate_model(model, test_loader, model_name):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.eval()
    all_preds, all_labels = [], []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device).unsqueeze(1)
            outputs = model(images)
            preds = (torch.sigmoid(outputs) > 0.5).float()
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    acc = accuracy_score(all_labels, all_preds)
    print(f"{model_name} Test Accuracy: {acc:.4f}")
    print(classification_report(all_labels, all_preds))
    return acc
```

In [15]:
```
# Evaluate both models
vit_test_acc = evaluate_model(vit_model, test_loader, "ViT")
swin_test_acc = evaluate_model(swin_model, test_loader, "Swin Transformer")
```

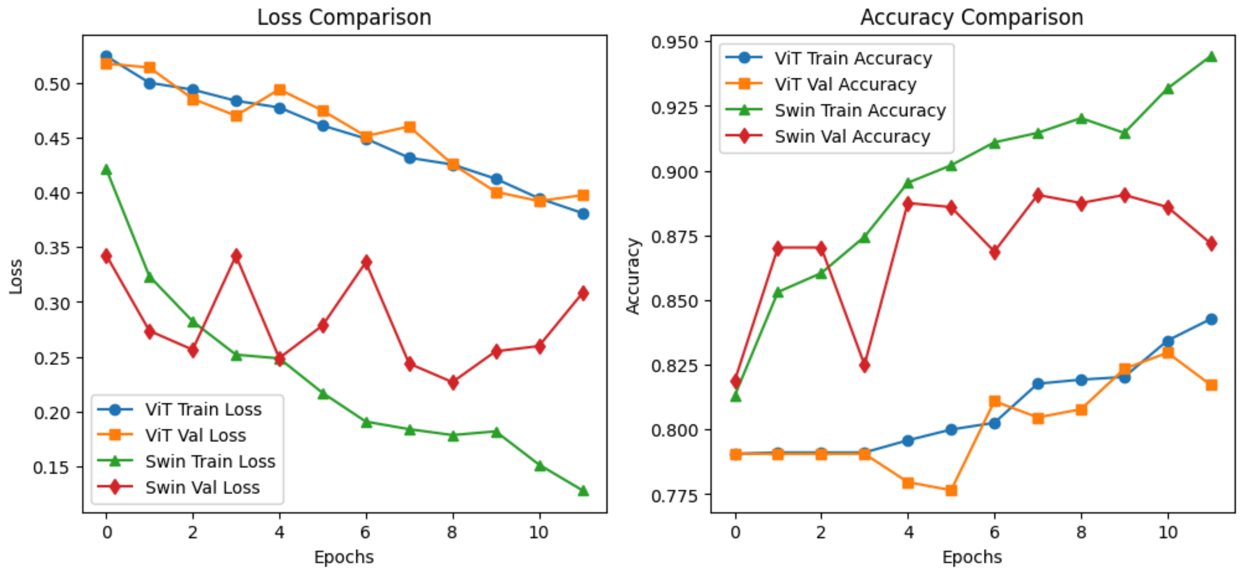
```
ViT Test Accuracy: 0.8422
              precision    recall  f1-score   support

         0.0       0.60      0.75      0.67       134
         1.0       0.93      0.87      0.90       506

    accuracy                           0.84       640
   macro avg       0.76      0.81      0.78       640
weighted avg       0.86      0.84      0.85       640

Swin Transformer Test Accuracy: 0.9328
              precision    recall  f1-score   support

         0.0       0.79      0.93      0.85       134
         1.0       0.98      0.93      0.96       506

    accuracy                           0.93       640
   macro avg       0.88      0.93      0.90       640
weighted avg       0.94      0.93      0.93       640
```

In [16]:
```
import matplotlib.pyplot as plt

# Debugging: Check the lengths of lists
print("Length of training/validation loss & accuracy lists:")
print("ViT Train Losses:", len(vit_train_losses))
```

```
print("ViT Val Losses:", len(vit_val_losses))
print("Swin Train Losses:", len(swin_train_losses))
print("Swin Val Losses:", len(swin_val_losses))

print("ViT Train Accs:", len(vit_train_accs))
print("ViT Val Accs:", len(vit_val_accs))
print("Swin Train Accs:", len(swin_train_accs))
print("Swin Val Accs:", len(swin_val_accs))

# Find the shortest epoch count (to avoid shape mismatch errors)
min_epochs = min(
    len(vit_train_losses), len(vit_val_losses),
    len(swin_train_losses), len(swin_val_losses),
    len(vit_train_accs), len(vit_val_accs),
    len(swin_train_accs), len(swin_val_accs)
)

# Trim all lists to match the shortest length
vit_train_losses = vit_train_losses[:min_epochs]
vit_val_losses = vit_val_losses[:min_epochs]
swin_train_losses = swin_train_losses[:min_epochs]
swin_val_losses = swin_val_losses[:min_epochs]

vit_train_accs = vit_train_accs[:min_epochs]
vit_val_accs = vit_val_accs[:min_epochs]
swin_train_accs = swin_train_accs[:min_epochs]
swin_val_accs = swin_val_accs[:min_epochs]

# Define the epoch range
epochs_range = range(min_epochs)

# Plot loss and accuracy comparison
plt.figure(figsize=(12, 5))

# Loss Plot
plt.subplot(1, 2, 1)
plt.plot(epochs_range, vit_train_losses, label="ViT Train Loss", marker="o")
plt.plot(epochs_range, vit_val_losses, label="ViT Val Loss", marker="s")
plt.plot(epochs_range, swin_train_losses, label="Swin Train Loss", marker="^")
plt.plot(epochs_range, swin_val_losses, label="Swin Val Loss", marker="d")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("Loss Comparison")

# Accuracy Plot
plt.subplot(1, 2, 2)
plt.plot(epochs_range, vit_train_accs, label="ViT Train Accuracy", marker="o")
plt.plot(epochs_range, vit_val_accs, label="ViT Val Accuracy", marker="s")
plt.plot(epochs_range, swin_train_accs, label="Swin Train Accuracy", marker="^
plt.plot(epochs_range, swin_val_accs, label="Swin Val Accuracy", marker="d")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
```

```
plt.legend()
plt.title("Accuracy Comparison")

plt.show()

# Print final test accuracy for both models
print(f"\nFinal Test Accuracy - ViT: {vit_test_acc:.4f}, Swin Transformer: {sw
```

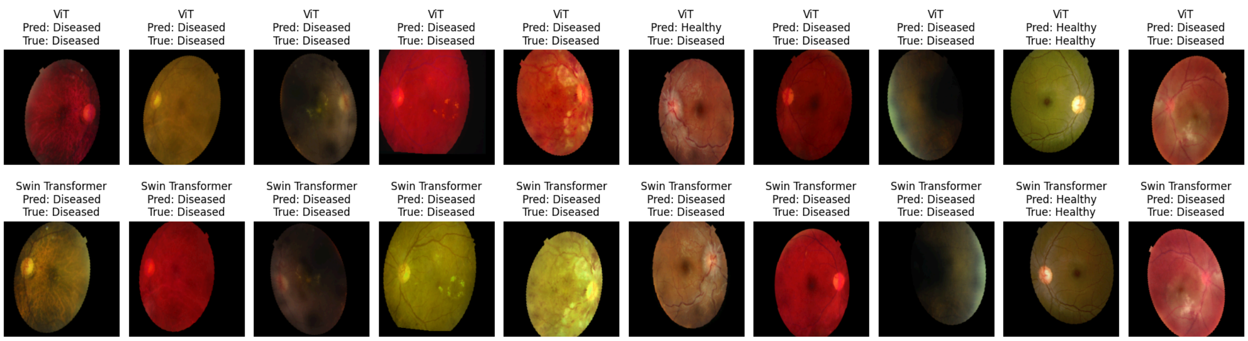
```
Length of training/validation loss & accuracy lists:
ViT Train Losses: 15
ViT Val Losses: 15
Swin Train Losses: 12
Swin Val Losses: 12
ViT Train Accs: 15
ViT Val Accs: 15
Swin Train Accs: 12
Swin Val Accs: 12
```




```
Final Test Accuracy - ViT: 0.8422, Swin Transformer: 0.9328
```

In [17]:
```
# Function to test the models on 10 images
def test_on_sample_images(models, test_dataset, model_names, num_samples=10):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    fig, axes = plt.subplots(2, num_samples, figsize=(20, 6))
    sample_indices = np.random.choice(len(test_dataset), num_samples, replace=

    for model, model_name, row in zip(models, model_names, range(2)):
        model.eval()
        with torch.no_grad():
            for i, idx in enumerate(sample_indices):
                image, label = test_dataset[idx]
                image = image.unsqueeze(0).to(device)
                output = model(image)
                pred = torch.sigmoid(output).item()
                predicted_label = "Diseased" if pred > 0.5 else "Healthy"

                image = image.squeeze(0).cpu().permute(1, 2, 0).numpy()
```

```
                image = (image * [0.229, 0.224, 0.225]) + [0.485, 0.456, 0.406
                image = np.clip(image, 0, 1)

                ax = axes[row, i]
                ax.imshow(image)
                ax.set_title(f"{model_name}\nPred: {predicted_label}\nTrue: {'
                ax.axis("off")

    plt.tight_layout()
    plt.show()

# Run the test function for both models
test_on_sample_images([vit_model, swin_model], test_dataset, ["ViT", "Swin Tra
```



In [26]:

```
torch.save(vit_model.state_dict(), "beit1.pth")
torch.save(swin_model.state_dict(), "swin1.pth")
```